Matthew Gardner

CS570

November 16th, 2024

Tayyaba Shaheen

Draft Paper – Artificial Intelligence Snake

This project implements three different versions of the classic game "Snake." The first version is the classical manual approach where the player controls the snake directly. The next two approaches are automated and require no player input. The second version uses an implementation of the A* search algorithm, and the third employs a Deep Q-Learning reinforcement learning technique. The learning model is rewarded for actions such as eating the fruit (increasing score) and penalized for collisions with obstacles or the snake's own body.

Through implementation, the two automation styles cannot match a skilled human player manually controlling the game. Both automation techniques can play the game at much faster speeds than a human, but when comparing average scores and highest scores, the human player wins by a large margin. In testing thousands of games, the A* implementation achieved an average score of 70 and a high score of 124. The learning model, however, is still in development and requires further optimization. Initial testing over 14,000 learning runs yielded an average score of 9 and a high score of 41. Researching similar implementations online, I struggled to find a learning approach that consistently compared to the A* algorithm. Published data often involved only 100 runs or so, making direct comparisons difficult.

This data suggests that while the A* algorithm could outperform an average snake player, it would likely struggle against a skilled player familiar with the game's intricacies and strategies.

Before the final submission, I plan to refine the learning model further and introduce more obstacles or variations to test the algorithms under different conditions.

Introduction:

As the field of artificial intelligence grows and grows every day in the technology industry and - even industry you it might not seem immediately applicable - I wanted to be a part of this boom by learning more about it. Specifically focusing on Artificial Intelligence in video games. The primary question being comparing these intelligences to the human intelligence that would normally be playing these games. For my implementation and testing I will be implementing two strategies into the classic game Snake.

If you're unfamiliar with Snake, the game is a very early videogame where you play as a snake that is constantly moving and only have the goal of eating as much fruit as possible. Once some fruit has been eating the snake grows in length, so as the game goes on it gets harder and harder and the snake grows and grows. The game ends if the player hits a wall our any part of its own body.

The first Artificial Intelligence approach implemented into the project is the A* search algorithm that will lead the snake to the fastest and safest path to the goal, or "fruit". This approach is very basic and struggles to deal with more complex situation such as changing states, for example in Snake the changing length of the snake and the risk of surrounding oneself in the walls or within its own body. The second implementation is a machine learning approach specifically a Deep Q-Learning reinforcement technique that involves rewarding the snake for taking certain actions like getting closer to the goal (fruit) and heavily rewarding the model for reaching the goal. The same approach is used as punishment for doing things you want to have the model avoid such

as colliding with the wall or its own body. This method requires a lot of time and effort to train the model as it can takes thousands of trainings runs to achieve even a small improvement, and in a game that has so many changing environments and states can be a struggled to train a consistent model to play the game at a high level.

In the following section of the player, we will explore previous work done by others, work that I have done so far, a discussion between my work and the work done by others, and a conclusion on the topic.

Research and Methodology:

From my initial research, the A* algorithm is most efficient in scenarios requiring the shortest and safest path to a goal. It combines Dijkstra's Algorithm, which explores vertices closest to the starting point, and Greedy Best-First Search, which prioritizes vertices closest to the goal using heuristic estimates. By combining these methods, A* achieves both efficiency and accuracy.

For the Snake implementation, the A* algorithm successfully directs the snake to the fruit until it becomes trapped. I found that adding a "coiling" behavior allowed the snake to wait safely for a new path to emerge, improving its performance. However, the algorithm remains unable to anticipate future consequences of its actions, often resulting in the snake getting trapped in a 1x1 section or a corner. Testing over thousands of runs showed that while A* achieved high scores, it faltered in later stages of the game.

In my literature review section of the project I did a lot of readings on learning models most of them on specifically a CNN (Convolutional Neural Network) but when starting the actual implementation again I found myself doing a lot of research and instead implementing the above mentioned Deep Q- Learning technique as it seemed far more efficient and easier to implement

specifically in python due to the torch library, which allowed for a much easier time implementing the learning model. Even though the technique implemented did change the main idea behind the AI is the same. the AI has all the same information that a normal person playing the game would have and can do all the same things a normal person could. The main difference is the time it takes for the AI to learn how to play the game and achieve a higher score. A quick note on what differs between this type of learning AI and normal AI, like what was discussed in the previous section: "The world is consumed with the machine learning revolution, and particularly the search for a functional artificial general intelligence, or AGI. Not to be confused with a conscious AI, AGI is a broader definition of machine intelligence that seeks to apply generalized methods of learning and knowledge to a broad range of tasks, much like the ability we have with our brains." (Lanham, p.8) The main difference is that the A* method is a much more methodical, mathematical approach, with heuristics and routing involved to find the fastest path. Whereas with a learning-based method, the AI just does what it wants every round, learning more and more about how to score higher points. From the reading *Hands-On Reinforcement Learning for Games*, there are four main elements to reward-based learning: the policy (representing the decisions and planning process of the agent), the reward function (the amount of reward an agent receives after completing a series of actions or an action), the value function (determining the value of a state over the long term), and finally the model (representing the environment in full, all game states).

For our specific Snake implementation, the A* implantation is very basic as it is simply just using the algorithm to find the best path to the goal and letting the snake follow that path, updating the path every time the snake moves. As I could not find much on a basic path finding implementation into Snake, this creates main goal of the project to compare this algorithm to a normal player and seeing how it reacts to the changing environment.

The learning model however is more complicated. For the rewarding of the model like discussed before we give the model a reward whenever it reaches and goal and give a negative reward whenever it has a collision. The main goal of this section of the project like the A* implementation is to compare the learning model to not only a human player but to the A* algorithm as well as the learning model takes much more time to get up and running.

Both models will also be put into different environments than just the normal state of the Snake game but an environment with preplaced obstacles, changing rules etc. to see how they react mostly the learning model after being trained in the normal state of Snake.

Current Work and Analysis:

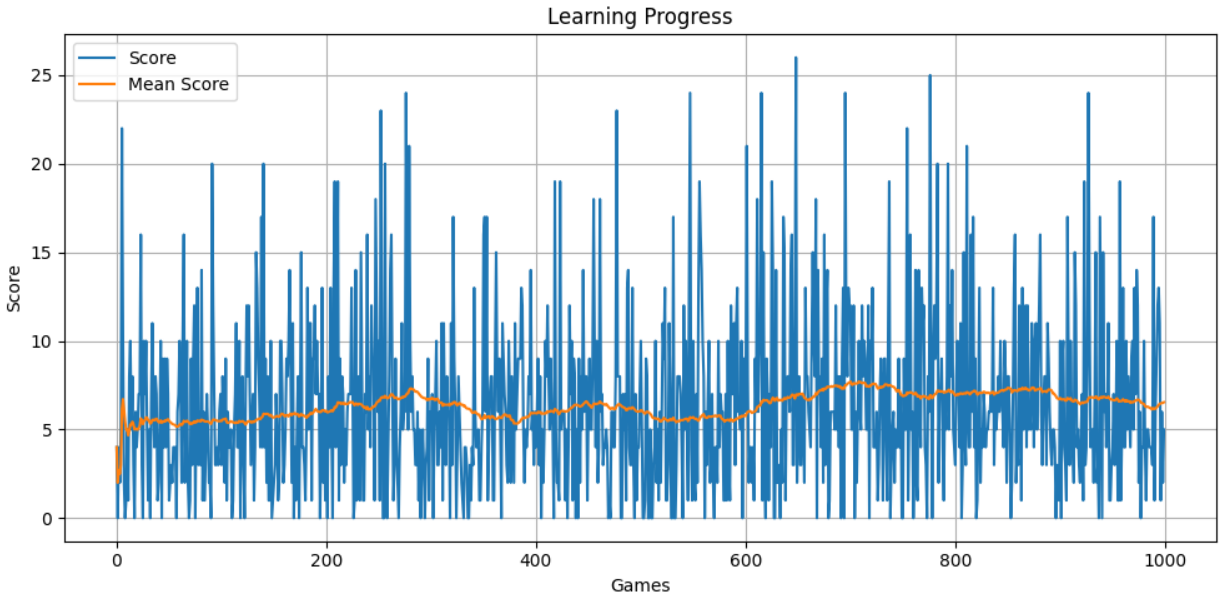
Current work done on the project, the project is around 70% done, all done in python The first step of the project was the game and the visuals both having been fully implemented. The game can be played normally by a person using the keyboard and can be seen working on the screen of the device. This is done fully through the python library PyGame, rendering everything to its own application screen on the computer. PyGame also handles things like updating the screen every frame, player controls/movement. When you launch the program through its main driver you will first be entered into a menu where you have three options, being Normal, A* mode, and learning mode. Selecting an option loads the game into those specific mode to be run one times or multiple by manually letting the game reset each time. Through the code however the game can be run in automation mode that is mostly used for training the model and collecting data letting the program run thousands of times without requiring any input.

Now discussing the first Artificial Intelligence implementation the A* algorithm, using the same logic for the base game we just create a function that is constantly being called in the update

function from PyGame to find the best path to the goal node which the program is also tracking. After implementing this I did however find that the main issue being that the snake would often find no path to the fruit due to specific situations such as the snake's body is blocking the fruit, the snake has entered a loop of its own body etc. to solve this issue I implemented another function that would have the snake coil itself whenever it was unable to find a path to hopefully allow the snakes bodies time to clear and the A* algorithm to find a safe path to the fruit. This implementation did improve the A* algorithms chances, but the algorithm still however could not see the consequences of its actions so on average the way this algorithm had a collision is sending the head of the snake into a situation where nothing it does can result in it continuing, such as running into a 1x1 section, running into a corner etc. After testing this algorithm 1000's of time it did achieve a reasonable high score of 124 and an average of 70, a score that many normal human players may struggle to achieve. This data however does show the weakness of this algorithm being later, higher score portions of the game.

The learning model was much more of a struggle than creating the game or implementing the A* algorithm and is still not fully complete. The Deep Q learning model is implemented using the python library Torch. The implementation is broken up into a couple parts being the Q network, and training modules, and the state of the board. To give the model the best chance both a long-term memory and short-term memory have been implemented with the table being updated every 100 runs. The information sent to the snake is very similar to what a human player would be able to receive just playing the game normally, being: the snakes head position, the position of the fruit, current movement direction of the snake, distance to the walls, and the distance to a collision with the body of the snake. The reward function is very basic, rewarding the snake 10 for eating a fruit, and taking 10 away whenever the snake dies. I originally had implemented a reward system for

getting closer to the fruit and punishment for going away from the fruit but found that after a while this just scared the model and confused it, so I removed it. I also messed around heavily with the values for eating fruit and having a collision but found 10 was best in both directions because if the death penalty was higher the snake would just stay in the middle of the board and do circles to not even attempting to eat the fruit. In my testing through the various methods and attempts I have had very bad luck and find that much more optimization or training is needed, overall, I have done around 20000 training runs for the models, with one model receiving around 14000 runs with an average score of around 6 and a high score of 36. My most recent attempt is shown in the graph below, with an average score of around 9 and a high score of 41:



As you can see in the graph, the mean score fluctuates between up and down and doesn't have a solid growth rate, and the scores themselves are completely random going from high all the way down to scores of zero somehow. Before I can start implementing special event and obstacles to the game, I need the trained model to be adequate to compete with the human model and A* algorithm.

Through the research portion of this project, I know that a properly trained and created learning model can compute with human scores, so I know that something in my code or the way I am training the model is heavily holding it back from what it could be. Discussing the A* algorithm though compared to the research was very informative as it showed the true power of these search algorithms, being able to control the finer details of the game meant I could test this algorithm to the limits and I was able to find that the speed at which this algorithm worked even with the padding I added on or poorly optimized code was amazing, as it was able to play the game at speeds my monitor wasn't even able to show properly.

Conclusion:

In its current state, this project highlights the challenges of creating AI for video games, particularly for dynamic and evolving environments like Snake. While the A* algorithm offers a straightforward and effective solution, it lacks the adaptability required for high-level play. Deep Q-Learning, though more versatile, demands extensive training and optimization to achieve competitive results.

This work underscores the difficulty and potential of advanced AI in gaming. With further development, these techniques could pave the way for more sophisticated and engaging applications in the future.

For the work left on the project mainly revolves around the learning model and getting it running properly so it can probably be tested in changing environments. The final step of the project is to test all three modes in these special environments and again collect data for each, format the data in easily readable graphs, and compare the results.